U.S. PATENT APPLICATION

FOR

SYNCHRONOUS INTERFACE TO ASYNCHRONOUS PROCESSES

Inventor(s): Taras Shkvarchuk
Alexander Lerner

Assignee: Grand Central Communications, Inc.

BEYER WEAVER & THOMAS, LLP
P.O. BOX 778
BERKELEY, CA 94704-0778

SYNCHRONOUS INTERFACE TO ASYNCHRONOUS PROCESSES

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority under 35 U.S.C. §120 from U.S. Patent Application No. 09/820,964 for SYSTEM AND METHOD FOR MAPPING OF SERVICES filed March 30, 2001, and U.S. Patent Application No. 09/820,966 for SYSTEM AND METHOD FOR ROUTING MESSAGES BETWEEN APPLICATIONS filed March 30, 2001, the entire disclosures of both of which are incorporated herein by reference for all purposes. The present application is also related to U.S. Patent Application No. 10/727,089 for APPARATUS AND METHODS FOR PROVISIONING SERVICES filed December 2, 2003, U.S. Patent Application No. 10/728,356 for APPARATUS AND METHODS FOR CORRELATING MESSAGES SENT BETWEEN SERVICES filed December 3, 2003, and U.S. Patent Application No. 10/742,513 for APPARATUS AND METHODS FOR MEDIATING MESSAGES filed December 19, 2003, the entire disclosures of all of which are incorporated herein by reference for all purposes.

BACKGROUND

[0002] The present invention relates to techniques for enabling communication between synchronous and asynchronous processes. More specifically, the present invention enables interaction between synchronous and asynchronous web services in a network environment.

[0003] The web service provider market is one of the fastest growing segments of the software industry. Web service providers make enterprise applications, such as human resources administration, recruiting, travel and expense management, sales force automation, and so on, available to customers over the web on a subscription basis, or for free. These enterprise applications are fully managed and hosted by the web service providers, which results in significant cost savings for enterprises using the web services and eliminates many of the issues requiring individual enterprise application integration (EAI) solutions.

[0004] Some web service providers merely host and manage third-party packaged software for their customers (i.e., "managed hosters"). Others build new applications from the ground up to take advantage of the benefits and cost-savings of the web service provider model. Web service providers enjoy the profit margins and operational scalability of consumer Web companies like eBay and Yahoo, while at the same time offering the feature sets of complex enterprise software applications such as PeopleSoft and Siebel.

[0005] Client applications can locate the web services using a UDDI (Universal Description, Discovery, and Integration) service, which is based on XML (eXtended Markup Language) and SOAP (Single Object Access Protocol). XML is a markup language for documents containing structured information, that is, the information and an indication of its role, and is a common way to provide information over wide area networks, such as the Internet. SOAP is an XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used in combination with a variety of protocols, but its most frequent use is in conjunction with HTTP (Hyper Text Transfer Protocol). Web service providers can register contact addresses and descriptions of the provided web services in a UDDI directory, and prospective clients can use the UDDI directory as a "phone book for web services," and look up web services that fit the clients' needs.

[0006] Web services are typically described in a Web Service Description Language (WSDL), which is an XML-based format. WSDL enables separation of the description of the abstract functionality offered by a web service from concrete details of a service description, such as "where" and "how" the functionality is offered. WSDL describes web services starting with the messages that are exchanged between the web service provider and a requestor. The messages are first described abstractly and are then bound to a concrete network protocol and message format. A message consists of a collection of typed data items. A message exchange between the web service provider and the requestor is referred to as an operation. A collection of operations is a portType. Collections of portTypes are grouped and referred to as a service. A web service represents an implementation of a service and contains a collection of ports, where each port is an implementation of a portType, which includes all the concrete details the requestor needs in order to interact with the web service.

[0007] Two main communication models exist for communication between a requestor and a web service: a synchronous communication model and an asynchronous communication model. In the synchronous communication model, a requestor communicates synchronously with a synchronous web service, that is, the requestor sends a request to the web service and waits until a response is returned. The requestor is prevented from performing any other operations until the response is received.

[0008] In the asynchronous communication model, on the other hand, a requestor communicates asynchronously with a web service, that is, the requestor sends a request to the web service, some time may pass, and a response is returned by the web service. The requestor may perform other operations while waiting for the response from the web service.

Currently, the two communication models cannot work in conjunction, and as a consequence there is no way for a synchronous requestor to communicate directly with an asynchronous web service, or for an asynchronous requestor to communicate directly with a synchronous web service.

SUMMARY

[0009] In general, in one aspect, the invention provides methods and apparatus, including computer program products, implementing and using techniques for enabling communication between synchronous and asynchronous processes. One or more web services are selectively accessed from a client machine over a network. A request for information is received from a client machine with a conversion engine. The request is received over a synchronous interface. The request is processed in the conversion engine, and the processed request is transmitted over an asynchronous interface from the conversion engine to at least one web service.

[0010] Advantageous implementations can include one or more of the following features. The network can be a local area network or a wide area network. A response to the processed request can be received from the at least one web service with the conversion engine over the asynchronous interface, be processed in the conversion engine, and be transmitted over the synchronous interface from the conversion engine to the client machine. Receiving a request can include blocking the client machine until a response to the received request has been obtained from the web service and delivered to the client machine, until an error message has been delivered to the client machine, or until a predetermined time period has passed. Processing the request can include receiving the request at a synchronous post interface, placing the request in a receive queue, routing the request to one or more delivery queues, and transferring the request from the delivery queues to one or more asynchronous push interfaces.

[0011] A confirmation can be received from the at least one web service over the asynchronous interface that the processed request has been received by the at least one web service. Transmitting the processed request can include pushing the processed request to the at least one web service over the asynchronous interface, or transmitting an available processed request to the at least one web service through the asynchronous interface in response to polling of the asynchronous interface by the at least one web service. Processing the request can include performing security management including authentication, authorization, security policy enforcement, decryption, or validation of digital signatures.

Processing the response can include receiving the response at an asynchronous post interface, placing the response in a receive queue, and routing the response to a delivery queue for the client machine.

[0012] A confirmation can be transmitted to the at least one web service over the asynchronous interface that the response has been received by the conversion engine. Transmitting the processed response can include pushing the processed response to the client machine over the synchronous interface.

[0013] In general, in one aspect, the invention provides a conversion engine. The conversion engine has a synchronous interface, an asynchronous interface, and a processing module. The synchronous receives a request from a client machine communicating synchronously with the conversion engine over a network and delivers a response to the request from the conversion engine to the client machine over the wide area network. The asynchronous interface delivers the received request from the conversion engine to one or more web services communicating asynchronously over the wide area network, and receives a response to the request from the one or more web services over the wide area network. The processing module converts a synchronous request into an asynchronous request, and

converts an asynchronous response into a synchronous response.

[0014] Advantageous implementations can include one or more of the following features. The conversion engine can include a routing module that routes a received request to one or more web services, and routes a received response to the request to the client machine. The conversion engine can include a policy directory storing policies for performing security management including one or more of: authentication, authorization, security policy enforcement, decryption, and validation of digital signatures. The conversion engine can include a web service directory containing information about available web services and their communication interfaces, for example, in the form of one or more web service description language files.

[0015] In general, in one aspect, the invention provides methods and apparatus, including computer program products, implementing and using techniques for

[0016] converting a first web service description language file describing synchronous operations for a web service into a second web service description language file describing asynchronous operations. A first web service description language file describing synchronous operations for a web service is provided to a conversion engine. The first web service description language file is translated in the conversion engine into a second web service description language file describing asynchronous operations. The second web

service description language file is provided to the client machine for further generation of client machine code.

[0017] Advantageous implementations can include one or more of the following features. Translating can include one or more of the following operations: translating a types part of the first web service description language file into a types part of the second web service description language file, translating a message part of the first web service description language file into a message part of the second web service description language file, translating a port type part of the first web service description language file into a port type part of the second web service description language file, translating a bindings part of the first web service description language file into a bindings part of the second web service description language file, and translating a service part of the first web service description language file into a service part of the second web service description language file.

[0018] Translating a types part can include preserving any data structures defined in the first web service description language file in the second web service description language file. Translating a types part can include adding an acknowledge element in the asynchronous web service description language file, the acknowledge element describing an acknowledgement that is returned when a request is asynchronously posted to the conversion engine by the client machine. The acknowledgement can include a correlation identifier, which can be a session identifier, a token, or a call identifier.

[0019] Translating a message part can include adding messages to the asynchronous web service description language file that are particular to asynchronous communication, including one or more of: a message for returning an acknowledgement response, a message for polling, a message for acknowledging a received request, and a message for acknowledging a response from a web service. The message for polling can include one or more of: a message for polling using a session identifier, a message for polling using a topic, and a message for polling using a token. Translating a port type part can include inserting a port type for asynchronous post operations and a port type for asynchronous poll operations into the second web service description language file. The port type can contain one or more of the following polling options: polling by session identifier, polling by topic, and polling by token.

[0020] Translating a bindings part can include inserting binding for a post port type, inserting a binding for a poll port type, and setting an encoding for messages that include the port types to reflect the encoding used by the conversion engine. Translating a service part can include adding an asynchronous post port with a first uniform resource locator addressing

the conversion engine, and an asynchronous poll port with a second uniform resource locator to the conversion engine. A template stored in the conversion engine can be used for translating at least part of the synchronous web service description language file into the asynchronous web service description language file.

[0021] In general, in one aspect, the invention provides methods and apparatus, including computer program products, implementing and using techniques for converting a first web service description language file describing asynchronous operations for a web service into a second web service description language file describing synchronous operations. A first web service description language file describing asynchronous operations for a web service is provided to a conversion engine. The first web service description language file is translated in the conversion engine into a second web service description language file describing synchronous operations. The second web service description language file is provided to the client machine for further generation of client machine code. The invention can be implemented to include one or more of the following advantages. Requestors and web services can communicate with each other regardless of which communication model they support. For example, a synchronous requestor can communicate directly with an asynchronous web service. Similarly, an asynchronous requestor can communicate directly with a synchronous web service. This enables requestors to access a wider range of web services, hosting services both specific to the requestors' respective enterprises, as well as common application services, such as doing a customer credit check, neither of which the requestors need to create nor maintain.

[0022] The ability to communicate with web services, irrespective of their respective communication models, facilitates information and services exchange between enterprises. For example, two enterprises can access the same web service network, where they have access to common information and services. This, in turn, facilitates inter-enterprise processes such as assuring that there are enough raw materials available to build a certain product. More enterprise application functions can be hosted by web services and accessed on a subscription basis, instead of within the requestor's own enterprise, which may lead to significant cost savings. WSDL files describing synchronous web services can be converted into asynchronous WSDL files that can be used by asynchronous requestors to communicate with the synchronous web services, without the requestors having to know that they are communicating with synchronous web services. Use of asynchronous services may ease implementation of parallel data processing.

[0023] The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

[0024] FIG. 1 shows a schematic diagram of a system in which the invention can be implemented.

[0025] FIG. 2 shows a schematic diagram of how a synchronous requestor communicates through an integration services network with an asynchronous web service.

[0026] FIG. 3 shows a schematic diagram of how an asynchronous requestor communicates through an integration services network with a synchronous web service.

[0027] Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0028] The invention will now be described by way of example with reference to specific implementations that are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific implementations, it will be understood that this description is not intended to limit the invention to the described implementations. On the contrary, this detailed description is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims.

[0029] FIG. 1 shows a schematic diagram of a system (100), in which the invention can be implemented. As can be seen in FIG. 1, the system (100) includes a requestor (102), or client, that uses a synchronous communication model to access web services, and a requestor (104), or client, that uses an asynchronous communication model to access web services. The system further includes a web service (108) that is configured to be accessed asynchronously, and a web service (110) that is configured to be accessed synchronously. It should be noted that only two requestors and two web services are shown for simplicity. In a typical situation, typically, many web services and many requestors are present.

[0030] The system (100) further includes an integration services network (106), for enabling the requestors (102; 104) to communicate with the web services (108; 110) irrespective of the communication methods supported by the individual requestors and web services. The integration services network (106) is accessible for the requestors (102; 104) and the web services (108; 110) over a local area network, such as an intranet, or over a wide

area network, such as the Internet. The integration services network (106) facilitates interoperability using a wide variety of web services technologies and standards including, for example, SOAP, WSDL, WS-Security, WS-Policy, and Business Process Execution Language (BPEL), and mediates the technology differences in data formats, communications protocols and business policies through a set of established and defined policies. The system (100) can be either an "open system" or a "closed system." In an open system, typically, the requestors (102; 104) and web services (108; 110) are owned and/or managed by entities distinct from each other and distinct form the owner of the integration services network (106). In the closed system, the requestors (102; 104) and web services (108; 110) are typically owned and/or managed by the owner of the integration services network (106). Combinations of open and closed system are also possible, in which there are some commonly owned and/or managed components, and some components that are owned and/or managed by different entities.

[0031] As can be seen in FIG. 1, the integration services network (106) contains a message (112) that is sent from a requestor (102; 104) to a web service (108; 110) through the integration services network (106). The message can, for example, be a request for information that is sent by a specific requestor (102) to a specific web service (108) that is hosted at a remote location.

[0032] In some implementations, the integration services network (106) can also specify which requestors (102; 104) have permission to access the web services (108; 110). For example, the integration services network (106) can provide security management including authentication, authorization and security policy enforcement at various points in a message's (112) lifecycle inside the integration services network (106), for example, when a message (112) is sent into the integration services network (106) from a requestor (102; 104), and when the message (112) is routed to its destination web service (108; 110). In one implementation, the policies needed for the integration services network (106) to perform security management operations are stored securely in a policy repository (116) in the integration services network (106). According to various implementations, the requestors (102; 104) and the providers of the web services (108; 110) need not be aware of the message format requirements of the party to which the message (112) is sent, nor of any message format translation taking place in the integration services network (106). A more detailed discussion of exemplary processes for mediating messages between the synchronous and asynchronous communication formats is provided below with reference to FIGs. 2 and 3.

[0033] In addition to providing mechanisms for mediating messages (112) sent between requestors (102; 104) and web services (108; 110), the integration services network (106) also preferably includes a web service directory (114) for storing various information regarding the web services (108; 110) that are accessible through the integration services network (106). Such information may include, for example, information similar to the UDDI information, which facilitates locating web services (108; 110) for requestors (102; 104), WSDL files for the different web services (108; 110), and policies that control which requestors (102; 104) and web services (108; 110) may interact, and how the interaction occurs. The integration services network (106) preferably also includes mechanisms for creating and combining web services (108; 110), registering requestors (102; 104) and their identifying information, and handling messages (112) routed between web services (108; 110) and/or requestors (102; 104). The directory (114) may be created from one or more databases stored on one or more memory devices on one or more computing platforms. Implementations of the integration services network (106) can also include components enabling complex routing and business process management, which both have been described in the above-referenced patent applications.

[0034] FIG. 1 has been described with reference to web services. However, it should be clear that the concepts described above can be applied to any type of "service," such as any computer application, process, entity, or device accessible to other applications, processes, entities, or devices through an interface such as an application programming interface (API), user interface, or Internet web user interface by any of a variety of protocols over a network within an entity or over the Internet. A service can also include multiple methods or applications on a single device or distributed across multiple devices.

[0035] Additional mechanisms and techniques for provisioning services and for enabling disparate entities to interact according to the invention are described in U.S. Patent Applications No. 09/820,964, 09/820,966, 10/727,089, 10/728,356 and 10/742,513 incorporated herein by reference above. Any of the mechanisms described in these referenced applications can easily be applied with the techniques described herein.

[0036] Two exemplary scenarios will now be described with reference to FIGs. 2 and 3. In the first scenario, which is illustrated in FIG. 2, an implementation is shown in which a requestor (102) that supports only synchronous communication sends a request to the integration services network (106), which forwards the request to and receives a response from a web service (108) that supports only asynchronous communication. In the second scenario, which is described in FIG. 3, an implementation is shown in which a requestor

(104) that supports only asynchronous communication sends a request to the integration services network (106), which forwards the request to and receives a response from a web service (110) that supports only synchronous communication. The scenarios in which a synchronous requestor (102) communicates with a synchronous web service (108), and in which an asynchronous requestor (104) communicates with an asynchronous web service (110), are well described in the art as well as in the above-referenced patent applications, and will therefore not be discussed herein. In one implementation, the operations that will be described below are performed by a conversion engine, which forms part of the integration services network (106) and is dedicated to performing operations for connecting synchronous requestors with asynchronous web services, and asynchronous requestors with synchronous web services, respectively.

[0037] As can be seen in FIG. 2, a request to obtain information from an asynchronous web service (108) is sent by a synchronous requestor (102). The request is posted to a synchronous post interface (202) in the integration services network (106). The synchronous post interface (202) can, for example, be a SOAP over HTTP interface, or a HTTP Get interface. It should be noted that such an interface can also be used for the integration services network's (106) asynchronous interfaces. The difference, as will be seen below, is that for a synchronous interface, the requestor (102) and integration services network (106) agree that the requestor (102) will block until the integration services network (106) synchronously returns a response. For an asynchronous interface, on the other hand, the requestor (102) and the integration services network (106) agree that the integration services network (106) will only synchronously return an acknowledgement, and then the integration services network (106) will later make a response available asynchronously. The format of the posted request is identical to a request that would otherwise be posted directly to a synchronous web service. The synchronous post interface (202) blocks the request and places the request in a receive queue (204). In one implementation the posting of the request establishes an HTTP session between the synchronous requestor (102) and the integration services network (106). The synchronous requestor now waits until a response is delivered back from the integration services network (106). The request is then retrieved from the receive queue (204) and routed (206) to one or more delivery queues (208). Typically there is one delivery queue (208) for each web service (108) to which the request should be sent. For simplicity, only one delivery queue (208) is shown in FIG. 2. In one implementation, the routing is performed by a message router in the integration services network (106), and the message router also performs a security policy and permission verification to verify that the

requestor (102) is allowed to send requests to the web service (108) in question, and that the request has the proper format. In some implementations, complex routing, for example, rule-based routing, as described in the above-referenced patent applications, can be used. The integration services network (106) then retrieves the request from the delivery queue and pushes (210) the request out to the web service (108) that is associated with the delivery queue. The push delivery of the request is similar to what would have occurred had the requestor (102) been an asynchronous client and invoked the web service (108) directly. The asynchronous web service (108) receives, acknowledges, and processes the request. It should be noted that in one implementation the asynchronous web service (108) does not return a response synchronously to the received request, since this would be interpreted only as a delivery acknowledgement and not a response. Any responses from the asynchronous web service (108) are instead posted separately to the integration services network (106), as will be seen below.

[0038] When the asynchronous web service (108) is done processing the request and has generated a response, the response is posted asynchronously to a post interface (212) in the integration services network (106). As was discussed above, this post interface (212) can, for example, be a SOAP over HTTP interface, or a HTTP Get interface. Just like when a request is sent, the format of the posted response is identical to a response that would otherwise be posted directly to an asynchronous requestor, so the asynchronous web service (108) does not have to be aware of that the requestor (102) is asking for a synchronous response. The response can either be posted to the network by the web service, which is also referred to as pushing the request, or the integration services network (106) can poll the asynchronous web service (108) for a response. If the posting was successful, the post interface (212) returns a confirmation to the web service (108), and places the request in a receive queue (214). If the posting is unsuccessful, the post interface (212) returns an error, and the web service (108) attempts to post the response again at a later time. After successful posting and delivery to the receive queue (214), the response is retrieved from the receive queue (214) and routed (216) to a delivery queue (218) for the synchronous requestor (102). The post interface (202) continuously polls the delivery queue (218) to determine whether any responses have been posted to the delivery queue (218). When a response is detected in the delivery queue (218), the response is returned to the synchronous requestor (102) and the process ends.

[0039] From the synchronous requestor's (102) side, it appears as if the asynchronous web service (108) is a synchronous web service, since the synchronous requestor is not aware of any operations that occur within the integration services network (106). All the

synchronous requestor (102) sees is a request being sent out, blocked, and a response returned after some time has passed. It should be noted that other implementations are also possible. For example, the asynchronous web service (108) could use a polling interface instead of the push interface (210) to accept requests.

[0040] Turning now to the second scenario, which is described in FIG. 3, an implementation is shown in which a requestor (104) that supports only asynchronous communication sends a request to the integration services network (106), which forwards the request to and receives a response from a web service (110) that supports only synchronous communication. As can be seen in FIG. 3, a request to obtain information from a synchronous web service (110) is sent by an asynchronous requestor (104). The request is posted to an asynchronous post interface (302) in the integration services network (106), such as a SOAP over HTTP interface, or a HTTP Get interface. The format of the posted request is identical to a request that would otherwise be posted directly to an asynchronous web service. If the posting was successful, the asynchronous post interface (302) returns a confirmation to the asynchronous requestor (104), and places the request in a receive queue (304). If the posting is unsuccessful, the asynchronous post interface (302) returns an error, and the requestor (104) attempts to post the request again at a later time. After successful posting and delivery to the receive queue (304), the request is retrieved from the receive queue (304) and routed (306) to one or more delivery queues (308). Again, typically there is one delivery queue (308) for each web service (108) to which the request should be sent, but for simplicity, only one delivery queue (308) is shown in FIG. 3. Also here, the routing can be performed by a message router in the integration services network (106), which may also perform a security policy and permission verification to verify that the requestor (104) is allowed to send requests to the web service (110), and ensure that the request has the proper format. The integration services network (106) then retrieves the request from the delivery queue and pushes (310) the request out to the web service (110) that is associated with the delivery queue (308). The push delivery of the request is much like as if the requestor (104) had invoked the web service (110) directly. The pushing of the request establishes an HTTP session between the synchronous web service (110) and the integration services network (106).

[0041] When the synchronous web service (110) is done processing the request and has generated a response, the response is synchronously returned to the push interface (312), for example, a SOAP over HTTP interface, or a HTTP Get interface, in the integration services network (106) on the same HTTP session that the request was delivered on, and the HTTP

session between the integration services network (106) and the synchronous web service (110) is terminated. Just like when a request is sent, the format of the response is identical to a response that would otherwise be returned directly to a synchronous requestor, so the synchronous web service (110) does not have to be aware of that the requestor (104) requires an asynchronous response. The returned response is delivered to a receive queue (314) and routed (316) to a delivery queue (318) for the asynchronous requestor (104).

[0042] The asynchronous requestor continuously polls (320) the delivery queue (318) to determine whether any responses have been posted to the delivery queue (318). When a response is detected in the delivery queue (318) through the polling operation (320), the response is returned to the asynchronous requestor (104), which acknowledges the returned response, and ends the process. From the asynchronous requestor's (104) side, it appears as if the synchronous web service (110) is an asynchronous web service. It should be noted that other implementations are also possible. For example, a push interface could be used instead of the poll interface (320) to supply responses to the requests to the asynchronous requestor (104).

[0043] As was described above, WSDL is an XML-based description of how to connect to and communicate with a particular web service. A WSDL description abstracts a particular service's various connection and messaging protocols into a neutral interface description and forms a key element of the UDDI directory's service discovery model. A WSDL file typically has five different parts: a types part, a message part, a port type part, a bindings part, and a service part. The types part is used to define data structures, the messages part is used to define one or more messages, the port type part is used to define how messages will be grouped together into operations, the bindings part is used to define encoding and placement of message parts, and the service part is a collection of ports consisting of a binding and a network address (or URL) for the web service. Many synchronous web services use WSDL to describe the web service. When a synchronous requestor would like to synchronously invoke a synchronous web service, the requestor can obtain the WSDL file describing the synchronous web service, bring the WSDL file into their computer system, which parses the WSDL file and generates code on the requestor's computer system that can properly invoke the synchronous web service. However, as was seen in FIG. 3, many requestors communicate asynchronously and can therefore not benefit from the WSDL files that are designed for synchronous communication.

[0044] The invention solves this problem by translating a WSDL file published by a synchronous web service (110) with the integration services network (106) into a WSDL file

that can be used by an asynchronous requestor. In one implementation, the translation of the WSDL file for a synchronous web service (110) occurs when the synchronous web service (110) registers in the directory (114) of the integration services network (106), or "on the fly," that is, the original WSDL file is stored in the integration services network (106) until a user indicates interest in using the synchronous web service (110), at which time a translated WSDL file is generated and distributed to the user. When a requestor indicates interest in using the synchronous web service (110), the integration services network (106) asks the requestor whether the requestor would like to use the web service (110) synchronously or asynchronously. If the requestor selects asynchronous use, the translated WSDL file is supplied to the requestor, where the translated WSDL file is used to generate the necessary client code. Otherwise, the original WSDL file published by the web service (110) is provided to the requestor. In one implementation, the original WSDL file may be modified such that the synchronous web service (110) is invoked through the integration services network (106), for example, by modifying the synchronous web service's URL in the WSDL to point to the address of the integration services network (106). Table 1 below shows an exemplary synchronous WSDL file, and Table 2 shows a corresponding translated WSDL file for asynchronous consumption, for an exemplary web service "WeatherService" that provides weather information. Line numbers have been added on the left hand side of Tables 1 and 2 in order to facilitate the explanation below, and do not form part of the original WSDL file or the translated WSDL file.

```
1) <?xml version="1.0"?>
2) <definitions name="WeatherService"
3)       targetNamespace="http://www.example.com/weather"
4)       xmlns:tns="http://www.example.com/weather"
5)       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6)       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7)       xmlns="http://schemas.xmlsoap.org/wsdl/">
8)       <types>
9)             <schema
10)                  xmlns="http://www.w3.org/2001/XMLSchema"
11)      targetNamespace="http://www.example.com/weather">
12)                  <complexType name="weatherReport">
13)                        <sequence>
```

```
14)                              <element name="time"
   type="xsd:dateTime"/>
15)                              <element name="temperature"
   type="xsd:int"/>
16)                              <element name="humidity"
   type="xsd:int"/>
17)                              <element name="wind"
   type="xsd:string"/>
18)                         </sequence>
19)                   </complexType>
20)             </schema>
21)       </types>
22)       <message name="getWeatherRequest">
23)             <part name="zipcode" type="xsd:string"/>
24)       </message>
25)       <message name="getWeatherResponse">
26)             <part name="return" type="tns:weatherReport"/>
27)       </message>
28)       <portType name="publicWeatherPort">
29)             <operation name="getWeather">
30)                   <input message="tns:getWeatherRequest"/>
31)                   <output message="tns:getWeatherResponse"/>
32)             </operation>
33)       </portType>
34)       <binding name="weatherSOAPBinding"
   type="tns:publicWeatherPort">
35)             <soap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
36)             <operation name="getWeather">
37)                   <soap:operation soapAction=""/>
38)                   <input>
39)                         <soap:body use="encoded"
   namespace=http://www.example.com/weather
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
40)                   </input>
41)                   <output>
42)                         <soap:body use="encoded"
   namespace=http://www.example.com/weather
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
43)                   </output>
44)             </operation>
```

```
45)       </binding>
46)       <service name="WeatherService">
47)             <port name="weatherSOAPPort"
   binding="tns:weatherSOAPBinding">
48)                   <soap:address
   location="http://services.example.com/soap/weather"/>
49)             </port>
50)       </service>
51) </definitions>
```

Table 1

[0045] As can be seen in Table 1, the original WSDL file contains a types part (lines 8-21), a message part (lines 22-27), a port type part (lines 28-33), a bindings part (lines 34-45), and a service part (lines 46-50).

[0046] The type part of the WSDL file defines a data structure called "WeatherReport" (line 12). The WeatherReport data structure contains four data elements: time, temperature, humidity, and wind (lines 13-18).

[0047] The message part of the WSDL file defines messages that will be sent to and from the web service. The first message is a request message, getWeatherRequest (line 22), which has one part containing a string zip code (line 23). The second message is a response message, getWeatherResponse (line 25) that will be sent back and has one part called "return" (line 26) of the type WeatherReport. Thus, the response will contain the four elements defined above in the WeatherReport data structure.

[0048] The port type part of the WSDL file defines how the messages will be grouped together. In this example, an operation, getWeather (line 29) consists of an input message getWeatherRequest (line 30) and an output message getWeatherResponse (line 31), which were both described above. In a real world scenario, there would typically be many different operations, each of which has one input message and one output message. However, it should be noted that the WSDL format also fully allows operations that have only an input message but no output message and vice versa.

[0049] The bindings part of the WSDL file defines the protocol binding for the port, i.e., the publicWeatherPort (line 34). In this example, the binding is a SOAP RPC style (line 35), which occurs over HTTP (line 35). No SOAP action is needed (line 37). The input and output use SOAP encoded style (lines 39 and 42), as required by RPC interactions.

[0050] Finally, the services part of the WSDL file defines how the web service is exposed to the outside world in terms of addressing. In this example, there is one port, weatherSOAPPort (line 47) with the address http://services.example.com/soap/ weather (line 48).

[0051] When the synchronous WSDL file in Table 1 is translated into an asynchronous format, the resulting file is the WSDL file shown in Table 2 below. It should be noted that the translated WSDL file in Table 2 is simply an example. Other similar formats and content are also possible.

```
1) <?xml version="1.0" encoding="UTF-8"?>
2) <definitions
3)       targetNamespace="http://www.example.com/weather"
4)       xmlns:gcPoll="http://grandcentral.com/schemas/poll/v1"
5)       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6)       xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
7)       xmlns:tns="http://www.example.com/weather"
8)       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9)       xmlns:gcPmark="http://grandcentral.com/schemas/postmark/v1"
10)      xmlns="http://schemas.xmlsoap.org/wsdl/">

11)      <types>
12)            <schema
13)      targetNamespace="http://www.example.com/weather"
14)                  xmlns="http://www.w3.org/2001/XMLSchema">
15)                  <complexType name="weatherReport">
16)                        <sequence>
17)                              <element name="time"
   type="xsd:dateTime"/>
18)                              <element name="temperature"
   type="xsd:int"/>
19)                              <element name="humidity"
   type="xsd:int"/>
20)                              <element name="wind"
   type="xsd:string"/>
21)                        </sequence>
22)                  </complexType>
23)            </schema>
24)            <xsd:schema
```

```
25)                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
26)                   targetNamespace=
   "http://grandcentral.com/schemas/postmark/v1">
27)                   <xsd:complexType name="postmarkType">
28)                         <xsd:sequence>
29)                               <xsd:element minOccurs="0"
   name="session" nillable="true" type="xsd:string"/>
30)                               <xsd:element minOccurs="0" name="token"
   nillable="true" type="xsd:string"/>
31)                               <xsd:element minOccurs="0" name="call"
   nillable="true" type="xsd:string"/>
32)                         </xsd:sequence>
33)                   </xsd:complexType>
34)             </xsd:schema>
35)       </types>
36)       <message name="getWeatherRequest">
37)             <part name="zipcode" type="xsd:string"/>
38)       </message>
39)       <message name="postmarkResponse">
40)             <part name="postmark" type="gcPmark:postmarkType"/>
41)       </message>
42)        <message name="getMessageBySession">
43)             <part name="session" type="xsd:string"/>
44)       </message>
45)       <message name="getMessageByTopic">
46)             <part name="topic" type="xsd:string"/>
47)       </message>
48)       <message name="getMessageByToken">
49)             <part name="token" type="xsd:string"/>
50)       </message>
51)       <message name="getWeatherResponse">
52)             <part name="return" type="tns:weatherReport"/>
53)       </message>
54)       <message name="acknowlegeRequest">
55)             <part name="token" type="xsd:string"/>
56)       </message>
57)       <message name="acknowlegeResponse">
58)       </message>
59)       <portType name="postpublicWeatherPort">
60)             <operation name="getWeather">
61)                   <input message="tns:getWeatherRequest"/>
```

```
62)                   <output message="tns:postmarkResponse"/>
63)             </operation>
64)       </portType>
65)       <portType name="pollpublicWeatherPort">
66)             <operation name="pollGetWeatherBySession">
67)                   <input message="tns:getMessageBySession"/>
68)                   <output message="tns:getWeatherResponse"/>
69)             </operation>
70)             <operation name="pollGetWeatherByTopic">
71)                   <input message="tns:getMessageByTopic"/>
72)                   <output message="tns:getWeatherResponse"/>
73)             </operation>
74)             <operation name="pollGetWeatherByToken">
75)                   <input message="tns:getMessageByToken"/>
76)                   <output message="tns:getWeatherResponse"/>
77)             </operation>
78)             <operation name="acknowledge">
79)                   <input message="tns:acknowlegeRequest"/>
80)                   <output message="tns:acknowlegeResponse"/>
81)             </operation>
82)       </portType>
83)       <binding name="PostWeatherSOAPBinding"
    type="tns:postpublicWeatherPort">
84)             <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
85)             <operation name="getWeather">
86)                   <soap:operation soapAction=""/>
87)                   <input>
88)                         <soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://www.example.com/weather"/>
89)                   </input>
90)                   <output>
91)                         <soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
92)       namespace="http://grandcentral.com/schemas/postmark/v1"/>
93)                   </output>
94)             </operation>
95)       </binding>
96)       <binding name="PollWeatherSOAPBinding"
    type="tns:pollpublicWeatherPort">
```

```
97)            <soap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
98)            <operation name="pollGetWeatherBySession">
99)                  <soap:operation soapAction=""/>
100)                 <input>
101)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
102)     namespace="http://www.example.com/weather"/>
103)                 </input>
104)                 <output>
105)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://www.example.com/weather"/>
106)                 </output>
107)           </operation>
108)           <operation name="pollGetWeatherByTopic">
109)                 <soap:operation soapAction=""/>
110)                 <input>
111)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
112)     namespace="http://www.example.com/weather"/>
113)                 </input>
114)                 <output>
115)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://www.example.com/weather"/>
116)                 </output>
117)           </operation>
118)           <operation name="pollGetWeatherByToken">
119)                 <soap:operation soapAction=""/>
120)                 <input>
121)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
122)     namespace="http://www.example.com/weather"/>
123)                 </input>
124)                 <output>
125)                       <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://www.example.com/weather"/>
126)                 </output>
127)           </operation>
```

```
128)            <operation name="acknowledge">
129)                  <soap:operation soapAction=""/>
130)                  <input>
131)                        <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
132)     namespace="http://grandcentral.com/schemas/poll/v1"/>
133)                  </input>
134)                  <output>
135)                        <soap:body use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
136)     namespace="http://grandcentral.com/schemas/poll/v1"/>
137)                  </output>
138)            </operation>
139)     </binding>
140)     <service name="WeatherService">
141)            <port name="postweatherSOAPPort"
   binding="tns:PostWeatherSOAPBinding">
142)                  <soap:address
   location="http://pop.grandcentral.com/post/soaprpc/example.com/weathe
   r"/>
143)            </port>
144)            <port name="pollweatherSOAPPort"
   binding="tns:PollWeatherSOAPBinding">
145)                  <soap:address
   location="http://pop.grandcentral.com/poll/soaprpc"/>
146)            </port>
147)     </service>
148) </definitions>
```

Table 2

[0052] Just like the synchronous WSDL file shown in Table 1, the translated asynchronous WSDL file contains a types part (lines 11-35), a message part (lines 36-58), a port type part (lines 59-82), a bindings part (lines 83-139), and a service part (lines 140-147).

[0053] In the types part, the WeatherReport data structure is preserved from the original WDSL file (lines 15-22). A "postmarkType" data structure has been added (lines 27-33), which describes a postmark that is returned when a request is asynchronously posted to the integration services network (106). The postmark is used during the polling described above

with reference to FIG. 2 and FIG. 3, so that the integration services network (106) knows which response to look for during the polling operation. Three identifiers are provided in the postmark: a session (line 29), a token (line 30), and a call (line 31).

[0054] As can be seen, the message part in the translated WSDL file in Table 2 contains more messages than the original WSDL file. Just like in the original WSDL file, there is a getWeatherRequest message, where a zip code can be submitted (lines 36-38). The next message, postmarkResponse, is a response message containing a postmark (lines 39-41). The next message, getMessageBySession, describes a way to poll using a session identifier (lines 42-44). getMessageByTopic, describes a way to poll using a topic (lines 45-47). getMessageByToken, describes a way to poll using a token (lines 48-50). getWeatherResponse (lines 51-53) is, just like in Table 1, a message that contains the response from the web service. Finally, there are a couple of additional messages for acknowledging successful retrieval of messages from the integration services network (106) (lines 54-58).

[0055] The port type part contains a post port (lines 59-64) and a poll port (lines 65-82), since both a post port and a poll port is needed for asynchronous operation, as described above with reference to FIG. 3. The post port has an operation called getWeather (line 60), and the input is, just like in the original WSDL file, getWeatherRequest (line 61), but instead of getting the getWeatherResponse, a postmark response is given back (line 62). The poll port contains three different polling options: polling by session identifier (lines 66-69), polling by topic (lines 70-73), and polling by token (lines 74-77). As was described above, both the token for the response message and session were returned in the postmark. The topic would potentially be included in the request that was sent to the web service, similar to a subject line of a message. The token guarantees that the exact message will be obtained. A session can contain multiple messages, so polling by session will return one or more of multiple active messages related to a particular session. A topic is similar to a session in that the topic can also be present in multiple messages, and that polling by topic may return one or more messages related to the particular topic. When polling by one of these methods, the getWeatherResponse is returned, that is, the getWeatherResponse message is now returned asynchronously from the getWeatherRequest message, whereas in the original WSDL file in Table 1, the response would have been returned synchronously. The acknowledge operation (lines 78-81) is a way for the web service to tell the integration services network (106) that the message has been received and can be deleted from the delivery queue in the integration services network.

[0056] The bindings section is similar to the original WSDL file. As the reader skilled in the art will realize, there are other bindings than SOAP that could be used, for example, an HTTP Get binding, and that SOAP is only used herein by way of example for illustrative purposes. Finally, the service part has two ports: a post port, postweatherSOAPPort (line 141), and a poll port, pollweatherSOAPPort (line 144), that each has their own URL.

[0057] As the reader skilled in the art will realize, a corresponding translation can be made from a WSDL for an asynchronous web service into a WSDL for a synchronous client. However, at the present time asynchronous web services are far less common than synchronous web services, so the vast majority of WSDL conversions would occur in the manner described above. In doing such a translation, however, it is necessary to know which combinations of asynchronous operations can be combined into a single synchronous operation. Thus, the translation requires either some convention on naming of operations, in order to be performed completely automatically, or some interaction from a user indicating how asynchronous operations pair up to form a synchronous operation. Such a mapping can, for example, be accomplished with a mapping tool with which a user can select operations out of rendered WSDL information to indicate pairing into synchronous operations. The mapping can, for example, be done at the time of registering the web service with the integration services network (106).

[0058] In both cases, the directory in the integration services network can, for example, store templates for the conversions or translations that are later modified for the specific instance of a needed conversion. For example, a template can contain the parts of Table 2 above that are independent of the original WSDL file, such as the postmark data structure and all the messages that define polling and acknowledging. The template can have sections in which parts of the original WSDL are substituted into the template. Templates, however, are just one approach, and there can be other equally useful approaches as well.

[0059] The invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Apparatus of the invention can be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor; and method steps of the invention can be performed by a programmable processor executing a program of instructions to perform functions of the invention by operating on input data and generating output. The invention can be implemented advantageously in one or more computer programs that are executable on a programmable system including at least one programmable processor coupled to receive data and instructions from, and to transmit data and instructions to, a data

storage system, at least one input device, and at least one output device. Each computer program can be implemented in a high-level procedural or object-oriented programming language, or in assembly or machine language if desired; and in any case, the language can be a compiled or interpreted language. Suitable processors include, by way of example, both general and special purpose microprocessors. Generally, a processor will receive instructions and data from a read-only memory and/or a random access memory. Generally, a computer will include one or more mass storage devices for storing data files; such devices include magnetic disks, such as internal hard disks and removable disks; magneto-optical disks; and optical disks. Storage devices suitable for tangibly embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, such as EPROM, EEPROM, and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM disks. Any of the foregoing can be supplemented by, or incorporated in, ASICs (application-specific integrated circuits).

[0060] To provide for interaction with a user, the invention can be implemented on a computer system having a display device such as a monitor or LCD screen for displaying information to the user and a keyboard and a pointing device such as a mouse or a trackball by which the user can provide input to the computer system. The computer system can be programmed to provide a graphical user interface through which computer programs interact with users.

[0061] A number of implementations of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, the operations of the integration services network can be performed either on the requestors' computer systems, or on the web service's, respectively, and similar results could be achieved. However, this would place larger requirements on the requestors and web services, respectively, which can be avoided with the approaches described above, where the requestors and web services do not need to have any information about the configuration of the respective partners with which they communicate. In this case, the integration services network (106) acts a shared service available on the public Internet to the requestors and web services. Accordingly, other embodiments are within the scope of the following claims.

What is claimed is: